

# CMSC 250: Recursively Defined Sets and Binary Trees

Justin Wyss-Gallifent

April 20, 2023

1	Recursively Defined Sets . . . . .	2
	1.1 Introduction . . . . .	2
	1.2 Examples . . . . .	2
2	Binary Trees . . . . .	4
	2.1 Introduction . . . . .	4
	2.2 Recursive Definition . . . . .	4
	2.3 Properties of Binary Trees . . . . .	5

# 1 Recursively Defined Sets

## 1.1 Introduction

Recursive definitions may be used not just to define sequences of numbers but to define sets.

The basic idea is that we create a set  $S$  as follows:

- (a) We put some collection of things in  $S$ .
- (b) We give one or more rules which say: If certain things are in  $S$ , then certain other things must be, too.

## 1.2 Examples

**Example 1.1.** We can build a set  $S$  of all the even numbers as follows:

- (a) We put the number 0 in  $S$ .
- (b) Whenever  $a \in S$  we also put  $a + 2$  and  $a - 2$  in  $S$ .

It's fairly obvious (but can be proved!) that this set is the set of all even numbers.

**Example 1.2.** Suppose we build a set  $S$  as follows:

- (a) We put the number 1 in  $S$ .
- (b) Whenever  $x \in S$  we also put  $2x$  in  $S$ .

It's fairly obvious (but can be proved!) that this set is the set of all powers of 2 greater than or equal to 1.

We don't have to just build sets of numbers. Here are two examples.

**Example 1.3.** Suppose we build a set  $S$  as follows:

- (a) We put the pair  $(0, 0)$  in  $S$ .
- (b) Whenever  $(a, b) \in S$  we also put  $(a, b + 1)$  and  $(a + 1, b + 1)$ , and  $(a + 2, b + 1)$  in  $S$ .

What else is in the set other than  $(0, 0)$ ?

Well since  $(0, 0) \in S$  we know that  $(0, 1), (1, 1), (2, 1) \in S$ . But then since  $(0, 1) \in S$  we know that  $(0, 2), (1, 2), (2, 2) \in S$ .

This goes on forever and it's not entirely clear what is and is not in the set. For example is  $(10, 10)$  in the set? How about  $(10, 20)$ ? How about  $(20, 10)$ ?

Here is one with strings.

**Example 1.4.** Suppose we build a set  $S$  as follows:

- (a) We put the string  $X$  in  $S$ .
- (b) Whenever a string  $s \in S$  we also put  $Xs$  and  $YsY$  in  $S$ .

What else is in the set other than  $X$ ?

Well since  $X \in S$  we know that  $XX, YXY \in S$ . But then since  $YXY \in S$  we know that  $XYXY, YYXY \in S$ .

This goes on forever and it's not entirely clear what is and is not in the set.

Here is a somewhat familiar example.

**Example 1.5.** Suppose  $p, q, r$  are the only statement variables we have. The set  $S$  of sentences in propositional logic involving these statement variables can be constructed this way:

- (a) We put  $p, q, r$  in  $S$ .
- (b) Then:
  - If  $X, Y \in S$  then we put  $(X \wedge Y)$  in  $S$ .
  - If  $X, Y \in S$  then we put  $(X \vee Y)$  in  $S$ .
  - If  $X \in S$  then we put  $(\sim X)$  in  $S$ .

So now we know that for example:

- $p, q \in S$  so  $(p \wedge q) \in S$ .
- $(p \wedge q) \in S$  so  $(\sim(p \wedge q)) \in S$ .
- $(\sim(p \wedge q)), r \in S$  so  $((\sim(p \wedge q)) \vee r) \in S$
- Etc...

This actually then allows us to construct all statements in propositional logic (or statements equivalent to them) using  $p, q,$  and  $r$ .

## 2 Binary Trees

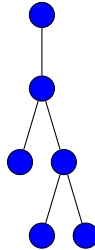
### 2.1 Introduction

Binary trees are another thing which can be built recursively.

Informally a binary tree starts with a root node. The root node can then be a parent node and connect down to either one or two child nodes. Each child node can then connect down to either one or two more child nodes. This keeps going until the tree ends at the leaf nodes, these are the child nodes without children of their own.

Formally speaking one way to define a binary tree is as a specific type of graph.

**Example 2.1.** Here is binary tree:





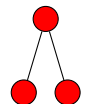

### 2.2 Recursive Definition

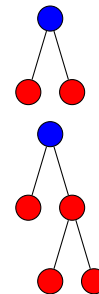
The set of binary trees  $B$  can also be defined recursively as follows:

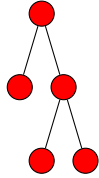
- (a) A single node is a binary tree, so a single node is in  $B$ .
- (b) If  $T_1$  and  $T_2$  are binary trees (are in  $B$ ) then we can create a new binary tree by taking a new node as the root and attaching  $T_1$  and  $T_2$  on branches below it.
- (c) If  $T_1$  is a binary tree (is in  $B$ ) then we can create a new binary tree by taking a new node as the root and attaching  $T_1$  on a branch below it.

According to this definition we get new binary trees as follows:

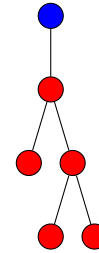
 and  are trees and therefore so is

 and  are trees and therefore so is





is a tree and therefore so is



(This last one is the first tree listed in the section.)

### 2.3 Properties of Binary Trees

Binary trees are heavily used in various algorithms and have several properties which are critical to know.

We'll just mention them here but we'll prove some of them via structural induction.

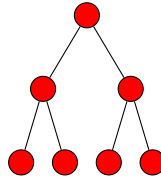
**Definition 2.3.1.** For a binary tree  $T$  define:

- $N(T)$  The number of nodes in the tree.
- $T(E)$  The number of edges (connections) in the tree.
- $L(T)$  The number of leaves in the tree.
- $H(T)$  The height of the tree. A single node has height 0.

In addition:

**Definition 2.3.2.** A binary tree is *perfect* if all the nodes (except the leaves) have exactly two children and if all the leaves are at exactly the same depth.

**Example 2.2.** Here is a perfect tree:



**Theorem 2.3.1.** For a binary tree  $T$  we have:

$$\begin{aligned}
 N(T) &= E(T) + 1 \\
 L(T) &\leq 2^{H(t)} && \text{With } = \text{ iff } T \text{ is perfect.} \\
 N(T) &\leq 2^{H(t)+1} - 1 && \text{With } = \text{ iff } T \text{ is perfect.}
 \end{aligned}$$

*Proof.* See the notes on structural induction.

*QED*

It's worth noting that as a result of the latter two we have the following:

**Theorem 2.3.2.** For a binary tree  $T$  we have:

$$\begin{array}{ll} H(t) \geq \lg L(t) & \text{With } = \text{ iff } T \text{ is perfect.} \\ H(t) \geq \lg(N(t) - 1) - 1 & \text{With } = \text{ iff } T \text{ is perfect.} \end{array}$$